NOKIA M2M PLATFORM
M2M SYSTEM PROTOCOL 2
SOCKET INTERFACE
USER MANUAL

NOKIA

Copyright © 2004 Nokia. All rights reserved. Issue 1.0 9312023

Contents

ACRONYMS AND TERMS 1
DEFINITIONS 2
1. ABOUT THIS DOCUMENT 3
2. INTRODUCTION 4
3. THE M2M SYSTEM PROTOCOL 2 SOFTWARE DESCRIPTION 6
3.1 SOCKET INTERFACE 6
3.2 NETWORK LINK LAYER 6
3.3 DATA LINK LAYER 6
3.4 PLATFORM WRAPPERS 7
4. INTEGRATING THE M2M SYSTEM PROTOCOL 2 WITH APPLICATION SOFTWARE 8
4.1 SOFTWARE IMPLEMENTATION 8
4.2 BUILDING THE SOFTWARE 9
4.3 PORTING THE SOFTWARE 9
5. THE M2M SYSTEM PROTOCOL 2 SOCKET INTERFACE 11
5.1 SOCKET API 11
5.1.1 Creating a socket (sp2_socket) 11
5.1.2 Binding a socket (sp2_bind) 12
5.1.3 Listening to a socket (sp2_listen) 13
5.1.4 Accepting a socket (sp2_accept) 15
5.1.5 Connecting a socket (sp2_connect) 16
5.1.6 Sending data to a socket (sp2_send / sp2_sendto) 17
5.1.7 Receiving data from a socket (sp2_receive) 19
5.1.8 Closing a socket (sp2_close) 20
5.1.9 Resolve host information (sp2_gethostbyname) 21
5.1.10 Network utility functions 22
5.2 M2M SYSTEM PROTOCOL 2 CONTROL API 22
5.2.1 Starting the M2M System Protocol 2 (sp2_start) 22
5.2.2 Stopping the M2M System Protocol 2 (sp2 _stop) 24
5.3 WIRELESS LINK CONTROL API 25
5.3.1 Opening a wireless link (sp2_link_open) 25

5.3.2 Closing a wireless link (sp2_link_close) 27
6. EXAMPLE OF M2M SYSTEM PROTOCOL 2 SOCKET USAGE 29
6.1 TYPICAL SOCKET USAGE 29
6.1.1 Using a client socket 29
6.1.2 Using a server socket 30
6.2 APPLICATION SOFTWARE EXAMPLE 31
6.2.1 Configuring and executing the Hello World application 32
6.2.2 The Hello World implementation 33
APPENDIX: NOKIA 12 GSM MODULE CONFIGURATION FOR THE HELLO WORLD APPLICATION 41

Legal Notice

Copyright © 2004 Nokia. All rights reserved.

Reproduction, transfer, distribution or storage of part or all of the contents in this document in any form without the prior written permission of Nokia is prohibited.

Nokia and Nokia Connecting People are registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Nokia operates a policy of continuous development. Nokia reserves the right to make changes and improvements to any of the products described in this document without prior notice.

Under no circumstances shall Nokia be responsible for any loss of data or income or any special, incidental, consequential or indirect damages howsoever caused.

The contents of this document are provided "as is". Except as required by applicable law, no warranties of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose, are made in relation to the accuracy, reliability or contents of this document. Nokia reserves the right to revise this document or withdraw it at any time without prior notice.

ACRONYMS AND TERMS

Acronym/term	Description
AM	Application Module
ANSI	American National Standards Institute
API	Application Programming Interface
CHAP	Challenge Handshake Authentication Protocol
DLL	Data Link Layer
IP	Internet Protocol
KB	Kilobyte (equivalent to 1024 bytes)
M2M	Machine-to-Machine (Man-to-Machine, Machine-to-Man)
NLL	Network Link Layer
OS	Operating System
OSI	Open Systems Interconnection
Platform	Operating system and communication services
PPP	Point-to-Point Protocol

DEFINITIONS

Term	Description
Socket API	M2M System Protocol 2 API for socket operations
M2M System Protocol 2 Control API	M2M System Protocol 2 API for serial link control
Wireless Link Control API	M2M System Protocol 2 API for GSM link control
Software wrapper	Used between software elements to isolate platform-specific features

1. ABOUT THIS DOCUMENT

This document describes, from application software point of view, how to integrate and use the M2M System Protocol 2. In addition to providing an overview of the M2M System Protocol 2 software, the document includes instructions on how to integrate the M2M System Protocol 2 with application software and how to use the APIs of the M2M System Protocol 2 socket interface. The document also includes an example of the M2M System Protocol 2 socket usage.

2. INTRODUCTION

The M2M System Protocol 2 software provides a socket-programming interface for application software. The socket interface may be used

- locally between Application Module (AM) applications
- to connect an AM application to a Java™ IMlet running on the Nokia M2M module
- to connect an AM application to machines on the Internet or intranet via wireless connectivity over the GSM network. The socket interface can also be used to connect an AM application to other remote machines that use IP.

Because the IP protocol stack placed at the disposal of the socket interface is located on the Nokia M2M module, application software must have a connection to the Nokia M2M module services. As illustrated in Figure 1, the M2M System Protocol 2 software is used to tunnel IP traffic between the AM and the Nokia M2M module via a serial connection.

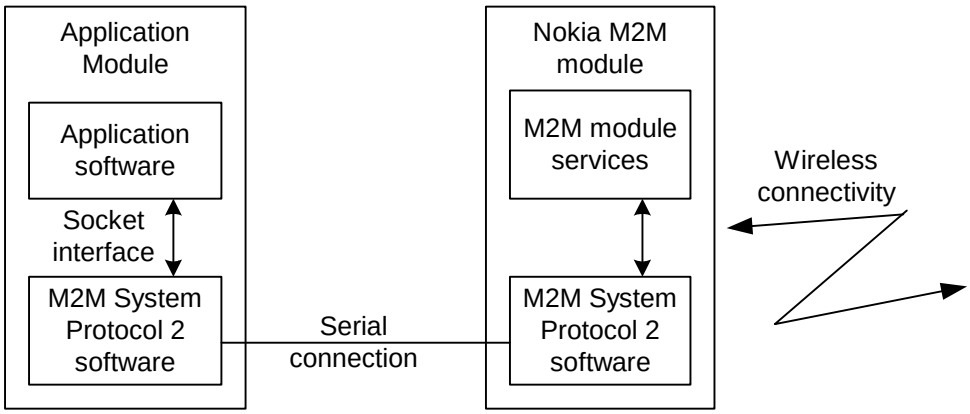

Figure 1. M2M System Protocol 2 overview

The socket interface provides a standard interface towards application software. In addition to socket handling, the socket interface makes it possible for the application to control both the wireless and serial connection. The M2M System Protocol 2 software connects the AM to the Nokia M2M module via a serial connection link, and enables the control of a M2M wireless connectivity link.

The serial connection is a physical connection between the AM and Nokia M2M module. It is connected between a serial port of the AM and the M2M System Connector of the Nokia M2M module. The serial connection defines the lowest layer transmission format using a point-to-point type of serial mode transmission as described in the *Nokia M2M Platform M2M System Protocol 2 Specification*.

The wireless connectivity makes it possible to connect the Nokia M2M module to a remote peer over a GSM network. Thus application software can be connected to sockets located on remote machines. For more information on the wireless connectivity solutions available via the Nokia M2M Platform, see the *Nokia M2M Platform Software Developer's Guide*.

3. THE M2M SYSTEM PROTOCOL 2 SOFTWARE DESCRIPTION

The M2M System Protocol 2 is based on the Open Systems Interconnection (OSI) Reference Model. It is a set of layers organized in a hierarchical structure, where each layer uses the layer below and provides services for the layer above. When data is sent from application software to a network, it flows downward the stack layer by layer until it has passed through the whole stack. When the data reaches the bottom layer it is passed to a network. The data received from the network flows the stack in an opposite direction. The software structure of the M2M System Protocol 2 is illustrated in Figure 2.

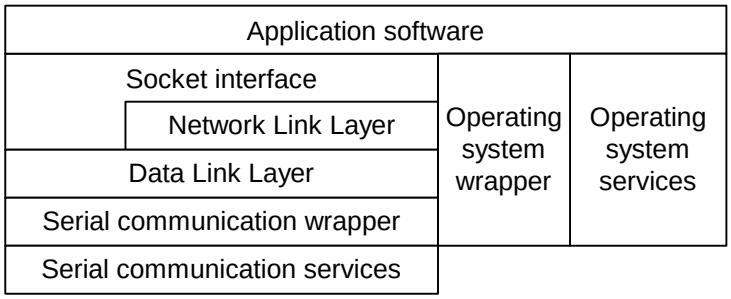

Figure 2. M2M System Protocol 2 software structure

3.1 SOCKET INTERFACE

The socket interface is on the top level of the M2M System Protocol 2. It provides all the services that application software uses for socket communication over the network. For more information on the socket interface, see Chapter 5.

3.2 NETWORK LINK LAYER

The Network Link Layer (NLL) is used to carry the M2M System Protocol 2 protocol primitives (requests/responses and indications/notifications) between the AM and the Nokia M2M module. For more information on the NLL, see the *Nokia M2M Platform M2M System Protocol 2 Specification*.

3.3 DATA LINK LAYER

The Data Link Layer (DLL) handles the transmission of protocol data packets between the AM and the Nokia M2M module over a serial connection. For more information on the DLL, see the *Nokia M2M Platform M2M System Protocol 2 Specification*.

3.4 PLATFORM WRAPPERS

The M2M System Protocol 2 is designed to operate in heterogeneous environments where a variety of operating systems and different types of serial communication hardware are used. The operating system and serial communication wrappers separate platform-specific functionality from the protocol stack in order to enhance the portability of the M2M System Protocol 2 implementation to different environments. For more information on the M2M System Protocol 2 platform wrapping, see Chapter 4.

4. INTEGRATING THE M2M SYSTEM PROTOCOL 2 WITH APPLICATION SOFTWARE

This chapter describes how the M2M System Protocol 2 is integrated as a part of application software.

4.1 SOFTWARE IMPLEMENTATION

The M2M System Protocol 2 is provided as ANSI-C implementation. The software is composed of the socket interface, protocol stack (NLL/DLL), and platform wrappers. Table 1 lists the header and source files of the M2M System Protocol 2 software.

Table 1. The M2M System Protocol 2 implementation files

Module	Header files	Source files
Socket interface	sp2_control_api.h sp2_socket_api.h sp2_link_api.h sp2_socket.h	sp2_control_api.c sp2_socket_api.c sp2_link_api.c
Protocol stack	sp2_nll.h sp2_dll.h sp2_rs.h sp2_port_socket.h	sp2_nll.c sp2_dll.c sp2_rs.c sp2_port_socket.c
Platform wrappers	sp2_system_common.h sp2_dbg_wrap.h sp2_port.h sp2_wait.h sp2_types.h	sp2_system_common_win32.c sp2_dbg_wrap_win32.c sp2_port_os_win32.c sp2_port_hw_win32.c

Application software uses the M2M System Protocol 2 functions via the socket interface. An application that uses the M2M System Protocol 2 functionality must include the header file `sp2_socket.h`.

The protocol stack provides the core functionality of the M2M System Protocol 2. It is recommended that application software is designed not to use the protocol stack directly.

The platform wrappers contain the platform-specific implementation of the operating system and serial communication services for the M2M System Protocol 2. The platform wrappers must be configured separately for each AM platform.

	Note: The M2M System Protocol 2 file names and external identifiers (except 'system' wrappers) are prefixed with 'sp2' to avoid name conflicts.

4.2 BUILDING THE SOFTWARE

This chapter provides a summary of how the M2M System Protocol 2 is built. The instructions are written with the assumption that the reader is already familiar with application software building process in general.

The M2M System Protocol 2 software building process includes the following steps:

1. Select platform wrapper source files for the M2M System Protocol 2.

   It is up to the application software developer to implement the AM platform wrappers if they are not available. For more information on porting the M2M System Protocol 2, see Chapter 4.3.

2. Compile all the M2M System Protocol 2 source files.

   Compile the selected platform wrappers and all the socket interface/protocol stack source files listed in Table 1. The socket interface and protocol stack source files should compile without modifications.

3. Link the compiled files to application software

   Once the M2M System Protocol source files are compiled (and optionally linked as a library), they are linked with application software to produce the final binary image.

	Tip: Chapter 6.2.2 includes a sample makefile that can be used to get started with building the M2M System Protocol 2 software.

4.3 PORTING THE SOFTWARE

As the M2M System Protocol 2 software is integrated with application software, the AM itself must have enough resources for both the M2M System Protocol 2 and application software. As an example, the resource requirements for the Win32 reference implementation of the M2M System Protocol 2 are listed in Table 2.

Table 2. M2M System Protocol 2 resource requirements

Resource	Requirement
Threads	SenderThread, ReceiverThread, MsgQueueThread
RAM	Static 4 KB (+ thread stack) + dynamic approx. 20 KB
ROM	Approx. 64 KB

When the M2M System Protocol 2 is started it will create the SenderThread, ReceiverThread, and MsgQueueThread. The SenderThread and ReceiverThread are high priority serial drivers used for packet sending and receiving, respectively. The MsgQueueThread takes care of serial message delivery to/from the M2M System Protocol protocol stack and its priority is application dependent.

The M2M System Protocol 2 memory requirements are approx. 64 KB ROM (without system libraries) and approx. 4 KB RAM. In addition, some memory is needed for dynamic memory and thread stacks. Dynamic memory size ranges from a few kilobytes to a maximum of 30 kilobytes; depending mainly on serial buffer sizes. The memory requirement for each thread stack is a few kilobytes.

Note: The *Nokia M2M Platform M2M System Protocol 2 Specification* also defines a minimum implementation for resource-constrained AM devices. The Nokia M2M module supports the minimum implementation.

5. THE M2M SYSTEM PROTOCOL 2 SOCKET INTERFACE

The socket interface is divided into the following APIs: Socket API, M2M System Protocol 2 Control API, and Wireless Link Control API. This chapter describes the APIs and lists the related methods, parameters, and return values. In addition, the chapter provides message sequence diagrams, which illustrate message exchange caused by API calls between the AM and Nokia M2M module.

For more detailed description of the messages, refer to the *Nokia M2M Platform M2M System Protocol 2 Specification*.

	Note: The M2M System Protocol 2 must be started before making any other M2M System Protocol 2 API calls. For more information on how to start the M2M System Protocol 2, see Chapter 5.2.1.

5.1 SOCKET API

The Socket API provides a standard socket interface. It supports client and server sockets, and UDP/TCP protocols.

5.1.1 Creating a socket (sp2_socket)

The `sp2_socket` function creates a handle to a socket on the AM side of the M2M System Protocol 2. The function call is illustrated in Figure 3.

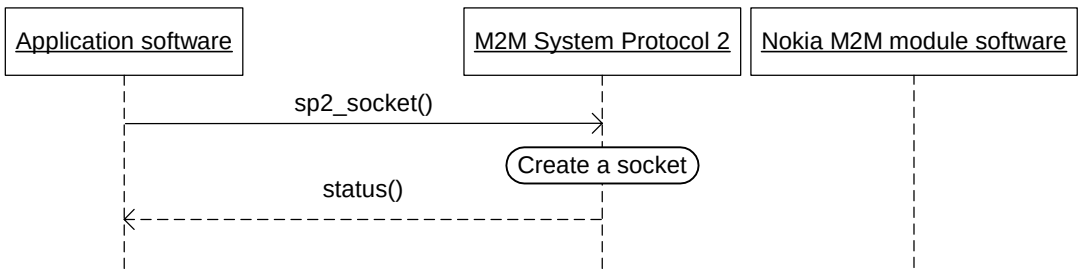

Figure 3. Creating a socket

The parameters and return values of the `sp2_socket` function call are described in Table 3 and Table 4.

Table 3. The sp2_socket function call parameters

Parameter	Description
Domain	IPv4

Parameter	Description
Type	Stream or datagram
Protocol	TCP or UDP

Table 4. The sp2_socket function call return values

Return value	Description
Socket ID	A handle to the created socket
SP2_SOCKET_ERROR	Socket allocation failed

An example of creating a socket:

```
DWORD socket; // a handle to the socket to be created
BYTE addrFormat = SP2_AF_INET; // addressing scheme is IPv4
BYTE type = SP2_SOCK_STREAM; // TCP stream socket
BYTE protocol = SP2_IPPROTO_TCP; // TCP/IP protocol is used

socket = sp2_socket(addrFormat,type,protocol);
if (socket != SP2_SOCKET_ERROR)
```

5.1.2 Binding a socket (sp2_bind)

The `sp2_bind` function binds a previously created M2M System Protocol 2 socket to a port. The function call is illustrated in Figure 4.

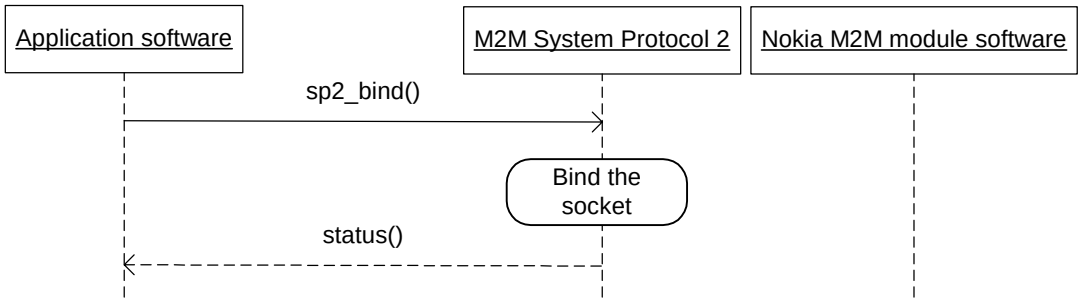

Figure 4. Binding a socket

The parameters and return values of the `sp2_bind` function call are described in Table 5 and Table 6.

Table 5. The sp2_bind function call parameters

Parameter	Description
Socket ID	A handle to a socket to be bound

Parameter	Description
Address	The address of the socket to be bound
Address length	The length of the socket address in bytes

Table 6. The sp2_bind function call return values

Return value	Description
SP2_SOCKET_OK	Socket was bound successfully
SP2_SOCKET_ERROR	Socket binding failed

An example of binding a socket:

```
DWORD socketID; // a handle to the socket
DWORD status; // a return value from the function
struct sp2_sockaddr_in addr; // socket address structure for IPv4

addr.sin_family = SP2_AF_INET; // addressing scheme is IPv4
addr.sin_port = sp2_htons(1234); // port to which the socket is to be bound
addr.sin_addr = sp2_inet_addr("127.0.0.1"); // a socket address

status = sp2_bind(socketID,(sp2_sockaddr *)&addr,sizeof(addr));
if (status == SP2_SOCKET_OK)
```

5.1.3 Listening to a socket (sp2_listen)

The `sp2_listen` function creates and binds a socket to the Nokia M2M module, and initiates the listening of the incoming socket connection requests. The function call is illustrated in Figure 5.

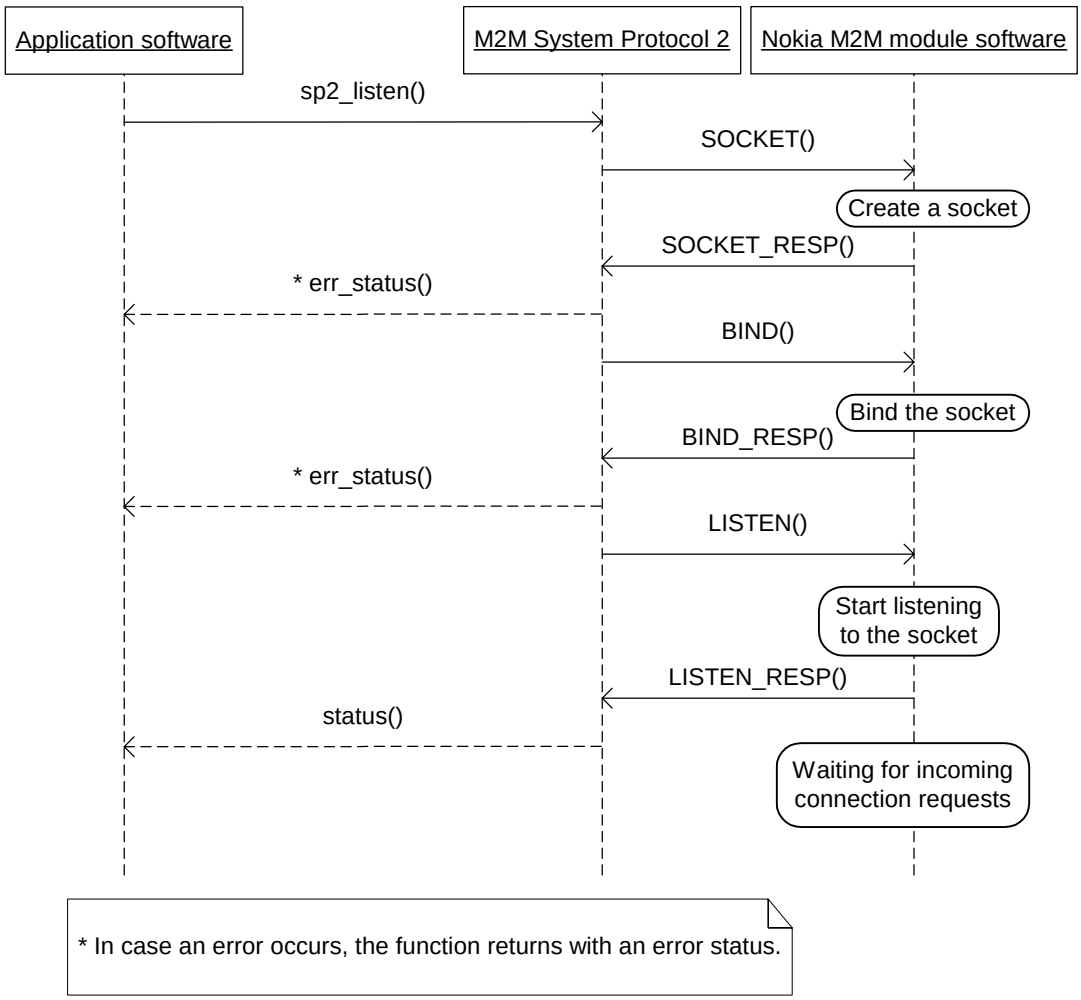

Figure 5. Listening to a socket

The parameters and return values of the `sp2_listen` function call are described in Table 7 and Table 8.

Table 7. The sp2_listen function call parameters

Parameter	Description
Socket ID	A handle to a socket to be listened
Queue length	Connection queue length: 0..5

Table 8. The sp2_listen function call return values

Return value	Description
SP2_SOCKET_OK	Socket was connected successfully

Return value	Description
SP2_SOCKET_ERROR	Connection failure

An example of listening to a socket:

```
DWORD socketID; // a handle to the socket
DWORD status; // a return value from the function
DWORD queueLen = 1; // queue length is 1

status = sp2_listen(socketID,queueLen);
if (status == SP2_SOCKET_OK)
```

5.1.4 Accepting a socket (sp2_accept)

The `sp2_accept` function accepts an incoming socket connection. The function will block until an ACCEPT_NOTIFICATION is received from the Nokia M2M module. The function call is illustrated in Figure 6.

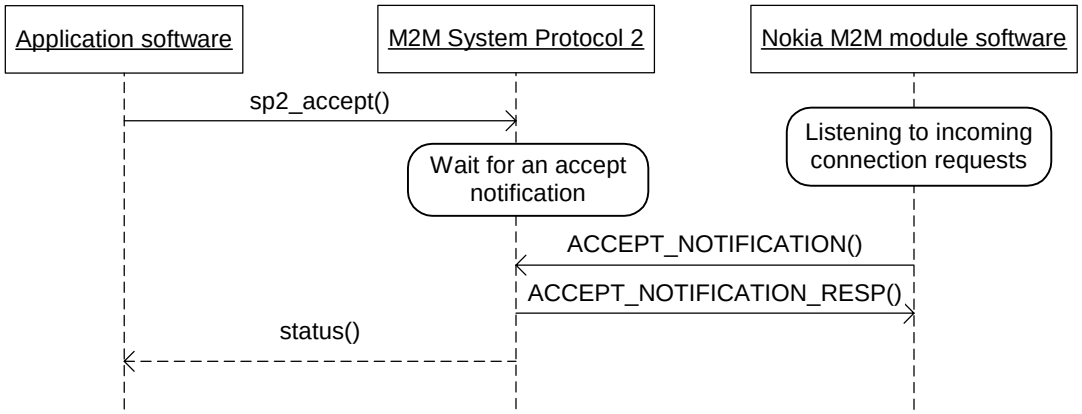

Figure 6. Accepting a socket

The parameters and return values of the `sp2_accept` function call are described in Table 9 and Table 10.

Table 9. The sp2_accept function call parameters

Parameter	Description
Socket ID	A handle to a socket which is listened to
Address	The address of the incoming socket
Address length	The length of the socket address in bytes

Table 10. The sp2_accept function call return values

Return value	Description
Socket ID	A handle to an incoming socket
SP2_SOCKET_ERROR	Acceptance failure

An example of accepting a socket:

```
DWORD socketID; // a handle to the socket
DWORD acceptedSocketID; // a handle to an incoming socket
struct sp2_sockaddr_in addr; // incoming socket address structure
DWORD addrlen; // incoming socket address length

acceptedSocketID = sp2_accept(socketID,(sp2_sockaddr *)&addr,&addrlen);
if (acceptedSocketID != SP2_SOCKET_ERROR)
```

5.1.5 Connecting a socket (sp2_connect)

The `sp2_connect` function connects a previously created M2M System Protocol 2 socket to another socket. The function call is illustrated in Figure 7.

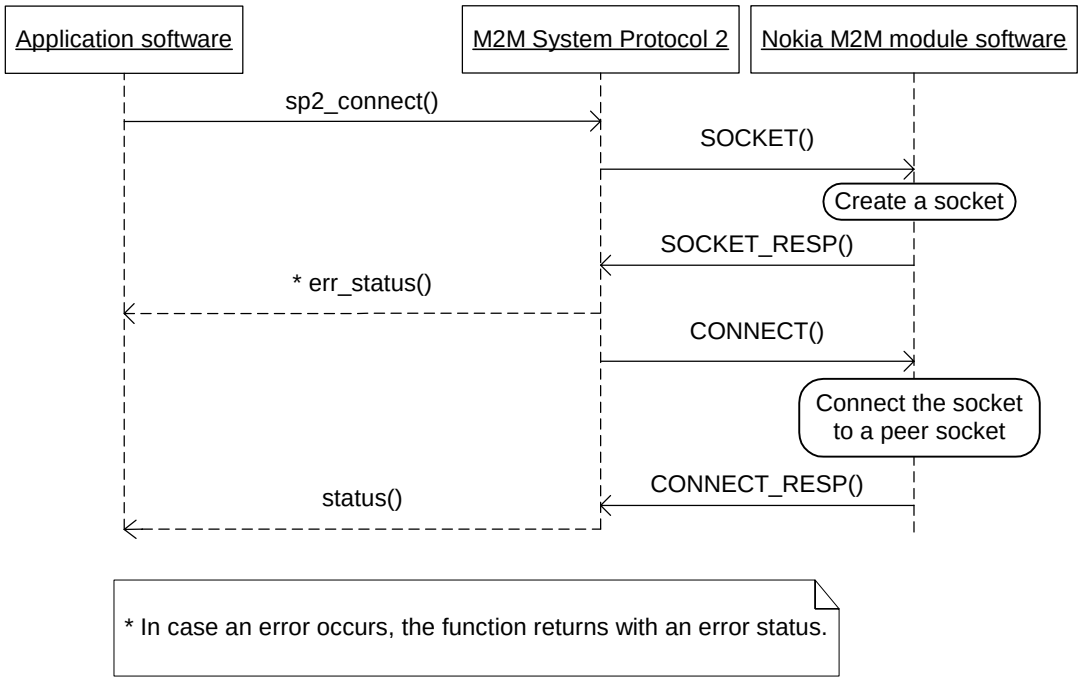

Figure 7. Connecting a socket

The parameters and return values of the `sp2_connect` function call are described in Table 11 and Table 12.

Table 11. The sp2_connect function call parameters

Parameter	Description
Socket ID	A handle to a socket to be connected
Address pointer	The address of a socket to which the existing socket is connected
Address length	The length of the socket address in bytes

Table 12. The sp2_connect function call return values

Return value	Description
SP2_SOCKET_OK	Socket was connected successfully
SP2_SOCKET_ERROR	Connection failure

An example of connecting a socket:

```
DWORD socketID; // a handle to the socket
DWORD status; // a return value from the function
struct sp2_sockaddr_in addr; // the address of the socket to be connected

addr.sin_family = SP2_AF_INET; // addressing scheme is IPv4
addr.sin_port = sp2_htons(1234); // port to which socket is to be connected
addr.sin_addr = sp2_inet_addr("127.0.0.1"); // a socket address

status = sp2_connect(socketID,(sp2_sockaddr *)&addr,sizeof(addr));
if (status ==  SP2_SOCKET_OK)
```

5.1.6 Sending data to a socket (sp2_send / sp2_sendto)

The `sp2_send` and `sp2_sendto` functions write data to a previously connected M2M System Protocol 2 socket. The `sp2_send` function is used with TCP sockets, and the `sp2_sendto` function is used with UDP sockets. Both function calls are illustrated in Figure 8.

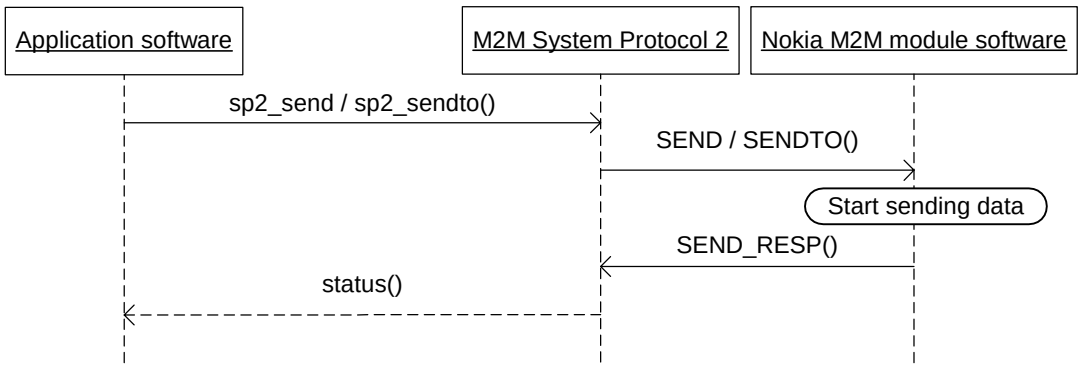

Figure 8. Sending data to a socket

The parameters of the `sp2_send` function call are described in Table 13.

Table 13. The sp2_send function call parameters

Parameter	Description
Socket ID	A handle to a socket to which data is written
Buffer	Data to be sent
Data length	The length of the data in bytes
Flags	Send flags

The parameters of the `sp2_sendto` function call are described in Table 14.

Table 14. The sp2_sendto function call parameters

Parameter	Description
Socket ID	A handle to the socket to which data is written
Buffer	Data to be sent
Data length	The length of the data in bytes
Flags	Send flags
Address	The address of the socket to which data is written
Address length	The length of the socket address in bytes

The return values of both the `sp2_send` and `sp2_sendto` function calls are described in Table 15.

Table 15. The sp2_send and sp2_sendto function call return values

Return value	Description
Bytes written	Number of bytes written successfully
SP2_SOCKET_ERROR	Failure to write data

An example of sending data to a socket:

```
DWORD socketID; // the handle to the socket to which data is written
BYTE buff[DATA_LENGTH]; // buffer to store data to be written
DWORD nbytes = DATA_LENGTH; // number of bytes to be written
DWORD flags = 0; // no flags
DWORD bytesWritten; // number of bytes written successfully

bytesWritten = sp2_send(socketID,buff,nbytes,flags);
if (bytesWritten != SP2_SOCKET_ERROR)
```

5.1.7 Receiving data from a socket (sp2_receive)

The `sp2_receive` function instructs the Nokia M2M module to read data from a socket. The `sp2_receive` will block and wait until data is available. The function call is illustrated in Figure 9.

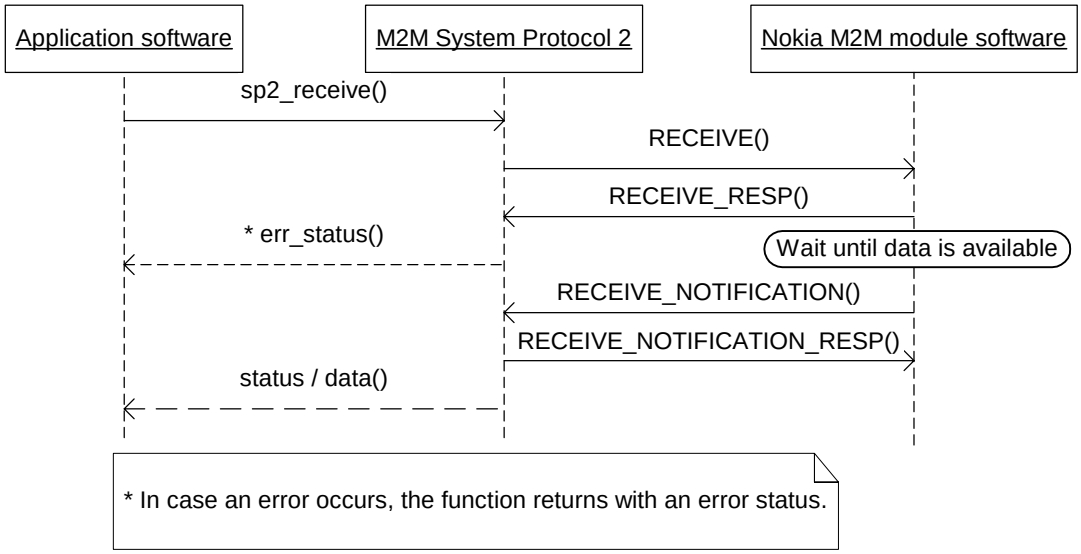

Figure 9. Receiving data from a socket

The parameters and return values of the `sp2_receive` function call are described in Table 16 and Table 17.

Table 16. The sp2_receive function call parameters

Parameter	Description
Socket ID	A handle to the socket from which data is read
Buffer	Buffer for the data to be read
Data length	The length of the data to be read in bytes
Flags	Receive flags

Table 17. The sp2_receive function call return values

Return value	Description
Bytes received	Number of bytes read successfully
SP2_SOCKET_ERROR	Failure to receive data

An example of receiving data from a socket:

```
DWORD socketID; // a handle to the socket
BYTE buff[DATA_LENGTH]; // buffer to store received data
DWORD nbytes = DATA_LENGTH; // number of bytes to be read
DWORD flags = 0; // no flags used
DWORD bytesRead; // number of bytes read successfully

bytesRead = sp2_receive(socketID,buff,nbytes,flags);
if (bytesRead != SP2_SOCKET_ERROR)
```

5.1.8 Closing a socket (sp2_close)

The `sp2_close` function releases a socket handle and instructs the Nokia M2M module to close a socket. The function call is described in Figure 10.

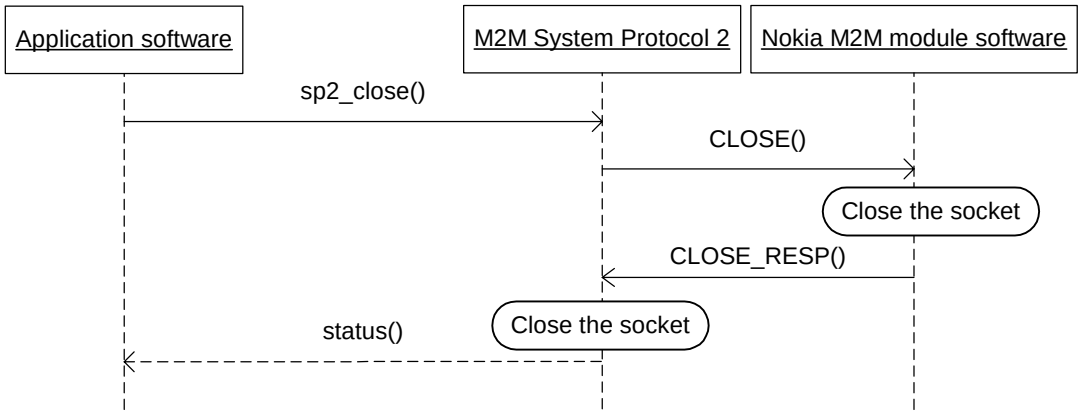

Figure 10. Closing a socket

The parameters and return values of the `sp2_close` function call are described in Table 18 and Table 19.

Table 18. The sp2_close function call parameters

Parameter	Description
Socket ID	A handle to the socket to be closed

Table 19. The sp2_close function call return values

Return value	Description
SP2_SOCKET_OK	Socket was closed successfully
SP2_SOCKET_ERROR	Failure to close a socket

An example of closing a socket:

```
DWORD socketID; // a handle to the socket
DWORD status; // a return value from the function

status = sp2_close(socketID);
if (status == SP2_SOCKET_OK)
```

5.1.9 Resolve host information (sp2_gethostbyname)

The `sp2_gethostbyname` function resolves host information based on domain name. The function call is illustrated in Figure 11.

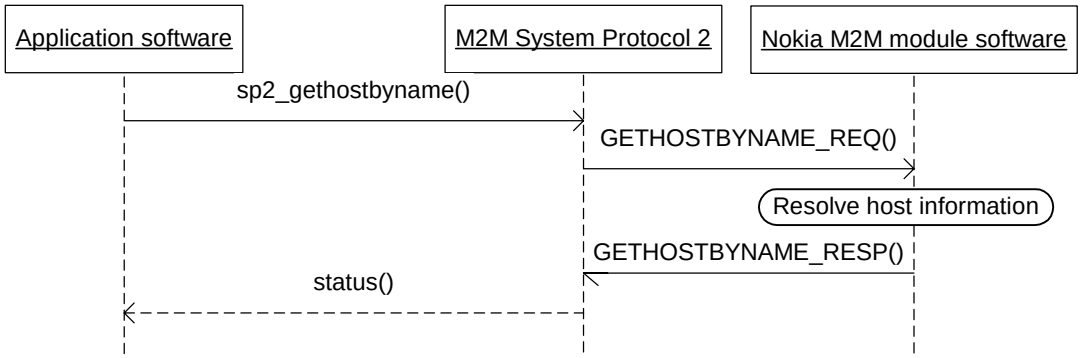

Figure 11. Resolving host information

The parameters and return values of the `sp2_gethostbyname` function call are described in Table 20 and Table 21.

Table 20. The sp2_gethostbyname function call parameters

Parameter	Description
String	A domain name to be resolved

Table 21. The sp2_gethostbyname function call return values

Return value	Description
Host information	Host information was resolved successfully
NULL	Failure to resolve a host

An example of resolving host information:

```
struct hostent *host; // function gethostbyname will reserve memory
const char *hostname = “local”; // a domain name

host = sp2_gethostbyname(hostname);
if (host != NULL)
```

```
{
  // use the host information and then release allocated memory
  free(host->h_name); // name string must be released by caller
  free(host); // host struct must be released by caller finally
}
```

5.1.10 Network utility functions

The M2M System Protocol 2 also provides some utility functions that are typically used when programming sockets. The utility functions are described below.

Convert a 32-bit value from host byte order to network byte order:

```
DWORD sp2_htonl(DWORD hostlong);
```

Convert a 16-bit value from host byte order to network byte order:

```
WORD sp2_htons(WORD hostshort);
```

Convert a 32-bit value from network byte order to host byte order:

```
DWORD sp2_ntohl(DWORD netlong);
```

Convert a 16-bit value from network byte order to host byte order:

```
WORD sp2_ntohs(WORD netshort);
```

Convert an IPv4 address from a dotted decimal string to 32-bit network byte ordered value:

```
DWORD sp2_inet_addr(const char *addr);
```

Convert a 32-bit network byte ordered IPv4 address to dotted decimal string:

```
char  *sp2_inet_ntoa(DWORD naddr);
```

5.2 M2M SYSTEM PROTOCOL 2 CONTROL API

The M2M System Protocol 2 Control API makes it possible to configure, open and close of a link between the AM and Nokia M2M module.

5.2.1 Starting the M2M System Protocol 2 (sp2_start)

The `sp2_start` function is used to establish an M2M System Protocol 2 link between the AM and Nokia M2M module.

	Note: The `sp2_start` function must be executed and the `sp2_hLinkReadyEvent` must be received before any other M2M System Protocol 2 API call can be made.

The `sp2_start` function configures the DLL according to the parameters given, and then starts the M2M System Protocol 2 threads. The function call is illustrated in Figure 12.

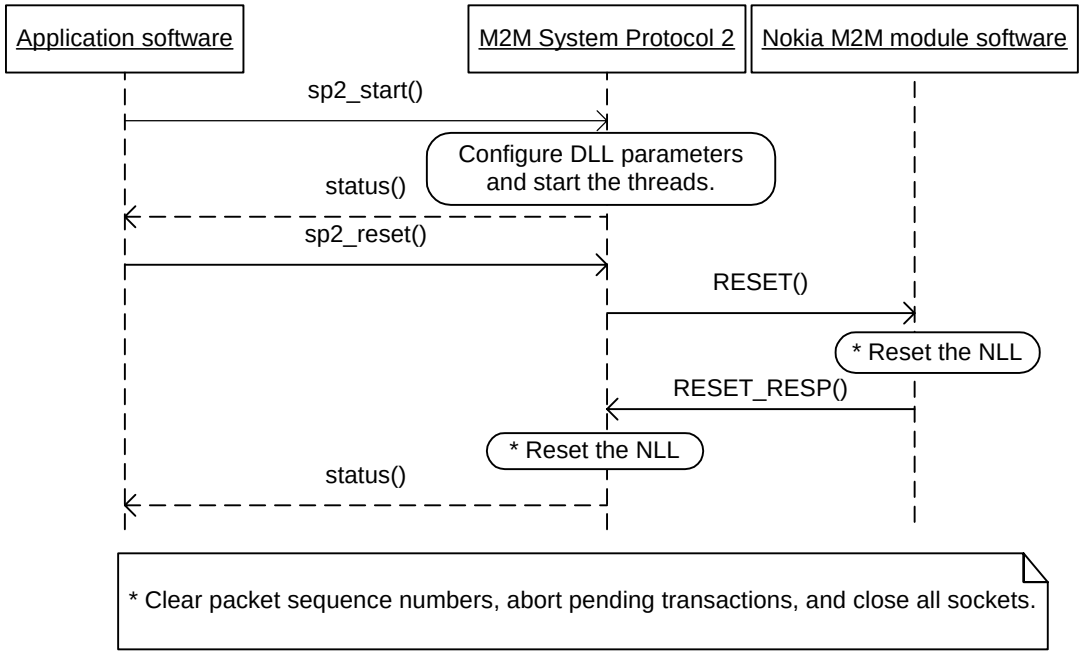

Figure 12. Starting the M2M System Protocol 2

The parameters and return values of the `sp2_start` function call are described in Table 22 and Table 23.

Table 22. The sp2_start function call parameters

Parameter	Description
N1	M2M System Protocol 2 parameter N1: 50..255
N3	M2M System Protocol 2 parameter N3: 5..10
T3	M2M System Protocol 2 parameter T3: 1..15s
Port	Serial port number used for data link connection
Baud rate	Baud rate (bit/s) used: 9600, 19200, 38400, 57600, 115200

	Note: Typically only the port and baud rate parameters are modified. The recommended values for the N1, N3, and T3 parameters are 255, 10, and 5, respectively. For more information on modifying the N1, N3, and T3 parameter values, see the *Nokia M2M Platform M2M System Protocol 2 Specification.*

Table 23. The sp2_start function call return values

Return value	Description
SP2_OK	M2M System Protocol 2 was started successfully
SP2_ERROR	Failure to start the M2M System Protocol 2

An example of starting a M2M System Protocol 2 link:

```
BYTE n1 = 255; // maximum data field length: n = 16*(N1+1)
BYTE n3 = 10; // activity timer is 10s
BYTE t3 = 5; // system protocol parameter T3
BYTE port = 1; // use COM port 1
DWORD baudrate = 115200; // serial speed is 115200 bps
DWORD status; // a return value from the function

status = sp2_start(n1,n3,t3,port,baudrate);
if (status == SP2_OK)
```

5.2.2 Stopping the M2M System Protocol 2 (sp2 _stop)

The `sp2_stop` function is used to stop an M2M System Protocol 2 link between the AM and Nokia M2M module. The function call is described in Figure 13.

	Note: After this function is called and the `stop` event is received, the M2M System Protocol 2 resources are freed, and the protocol must be restarted before issuing any further API requests.

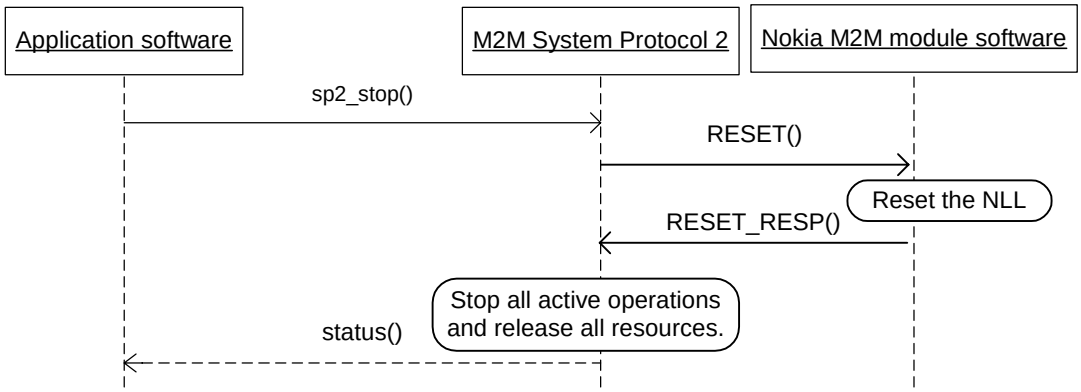

Figure 13. Stopping the M2M System Protocol 2

The `sp2_stop` function call does not have any parameters. The return values are described in Table 24.

Table 24. The sp2_stop function call return values

Return value	Description
SP2_OK	Link was stopped successfully
SP2_ERROR	Failure to stop a link

An example of stopping a M2M System Protocol 2 link:

```
DWORD status; // a return value from the function

status = sp2_stop();
if (status == SP2_OK)
```

5.3 WIRELESS LINK CONTROL API

The Wireless Link Control API makes it possible to open and close a pre-configured connection of the Nokia M2M module. Since the configuration cannot be changed via the Wireless Link Control API, the connection configuration must be done either locally using the Configurator software for the Nokia M2M module or remotely over-the-air via the Nokia M2M Gateway.

5.3.1 Opening a wireless link (sp2_link_open)

The `sp2_link_open` function is used to establish a wireless link over a GSM network via the Nokia M2M module. The `sp2_link_open` function call is illustrated in Figure 14.

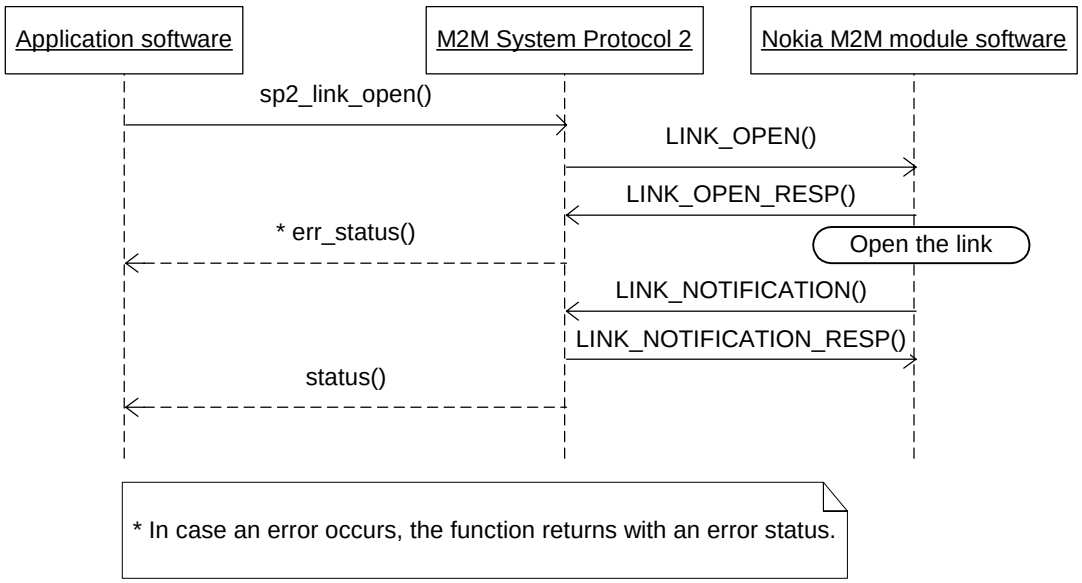

Figure 14. Opening a wireless link

The parameters of the `sp2_link_open` function call are described in Table 25. The return values from the `sp2_link_open` function call are described in Table 26.

Table 25. The sp2_link_open function call parameters

Parameter	Description
Connection ID	The Nokia M2M module connection ID to be checked and opened.

Table 26. The sp2_link_open function call return values

Return value	Description
Link ID	Handle to an opened link
SP2_LINK_ERROR	Failure to open a link

An example of opening a wireless link:

```
DWORD connectionID = SP2_DEFAULT_CONNECTION; // preconfigured connection ID
DWORD linkID; // a handle to the link that was opened

linkID = sp2_link_open(connectionID);
if (linkID != SP2_LINK_ERROR)
```

5.3.2 Closing a wireless link (sp2_link_close)

The `sp2_link_close` function is used to close a wireless link over a GSM network via the Nokia M2M module. The function call is depicted in Figure 15.

	Note: All remote sockets must be closed before closing the wireless link.

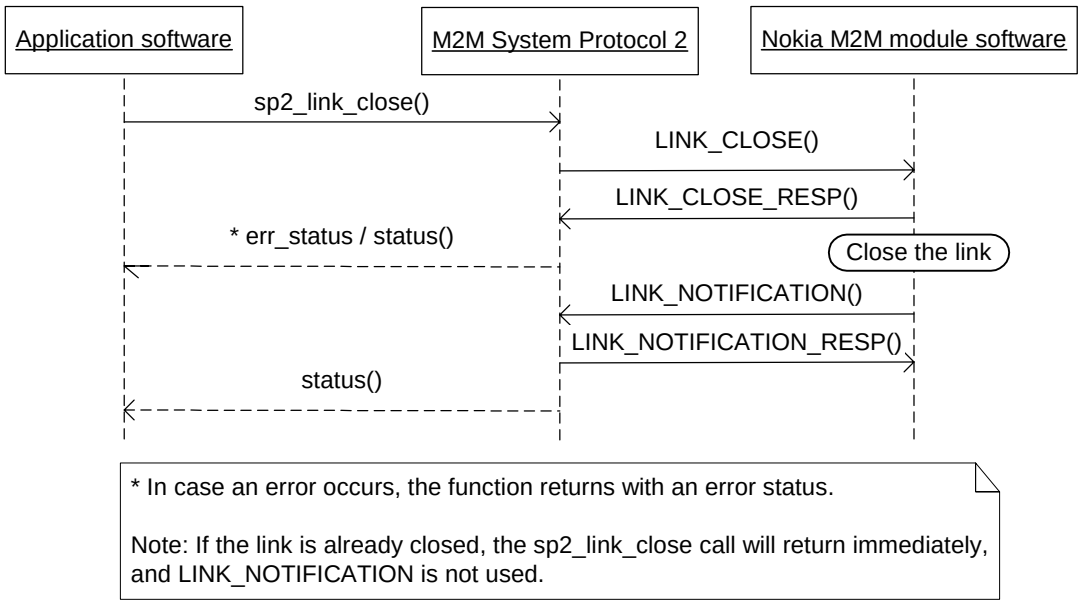
Application software
M2M System Protocol 2
Nokia M2M module software
sp2_link_close()
LINK_CLOSE()
LINK_CLOSE_RESP()
* err_status / status()
Close the link
LINK_NOTIFICATION()
LINK_NOTIFICATION_RESP()
status()
* In case an error occurs, the function returns with an error status.
Note: If the link is already closed, the sp2_link_close call will return immediately, and LINK_NOTIFICATION is not used.

Figure 15. Closing a wireless link

The parameters and return values of the `sp2_link_close` function call are described in Table 27 and Table 28.

Table 27. The sp2_link_close function call parameters

Parameter	Description
Link ID	Link ID to be closed

Table 28. The sp2_link_close function call return values

Return value	Description
SP2_LINK_OK	Link was closed successfully
SP2_LINK_ERROR	Failure to close a link

An example of closing a wireless link:

```
DWORD linkID; // a handle to the link

if (sp2_link_close(linkID) == SP2_LINK_OK)
```

6. EXAMPLE OF M2M SYSTEM PROTOCOL 2 SOCKET USAGE

6.1 TYPICAL SOCKET USAGE

Sockets are typically used for client-server communication. This chapter describes the typical use of the M2M System Protocol 2 client and server sockets. Although the sequences are illustrated from an individual socket's point of view, one M2M System Protocol 2 session and wireless GSM link may, of course, be used for many different sockets.

6.1.1 Using a client socket

Typically application software that uses a M2M System Protocol 2 client socket over wireless connectivity has to:

1. Open a serial link to the Nokia M2M module via the M2M System Protocol 2 Control API.
2. Reset the protocol via the M2M System Protocol 2 Control API.
3. Open a wireless GSM link via the Wireless Link Control API.
4. Communicate with the server application via the Socket API.
5. Close the wireless link via the Wireless Link Control API
6. Close the serial link to the Nokia M2M module via the M2M System Protocol 2 Control API.

A typical client socket message sequence is illustrated in Figure 16.

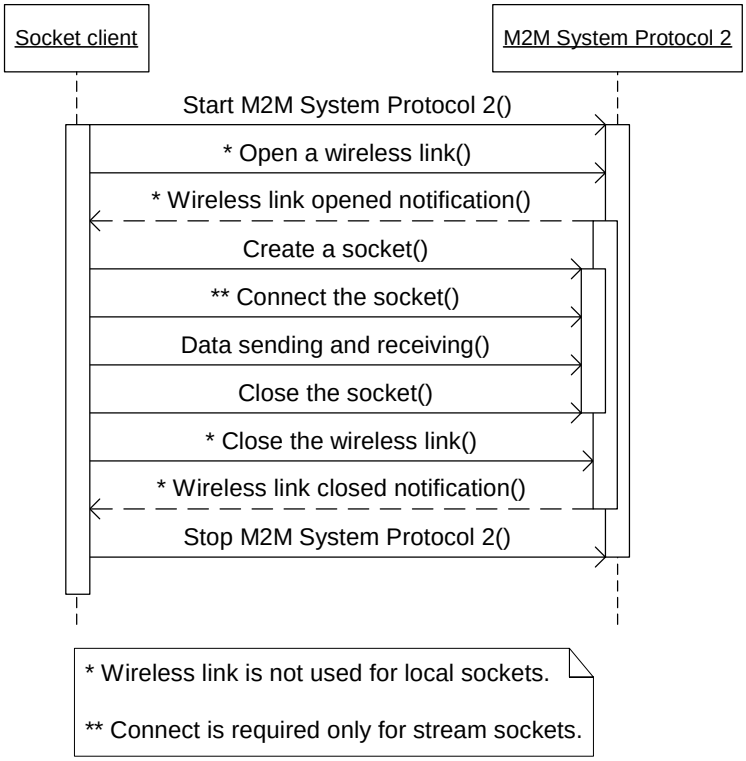
Socket client
M2M System Protocol 2
Start M2M System Protocol 2()
* Open a wireless link()
* Wireless link opened notification()
Create a socket()
** Connect the socket()
Data sending and receiving()
Close the socket()
* Close the wireless link()
* Wireless link closed notification()
Stop M2M System Protocol 2()
* Wireless link is not used for local sockets.
** Connect is required only for stream sockets.

Figure 16. A typical client socket message sequence

	Note: In case the wireless link breaks all new and pending socket functions using the link in question will fail. To execute the socket operations that were interrupted, the wireless link has to be re-established and the sockets have to be closed and recreated.

6.1.2 Using a server socket

Unlike with client sockets, server sockets do not need to control wireless GSM links. Instead, the Nokia M2M module handles wireless links automatically.

Typically application software that uses a M2M System Protocol 2 server socket has to:

1. Open a serial link to the Nokia M2M module via the M2M System Protocol 2 Control API.
2. Reset the M2M System Protocol 2 via the M2M System Protocol 2 Control API.
3. Wait for an incoming socket request via the Socket API.

4. Communicate with the client via the Socket API.

5. Close the serial link to the Nokia M2M module via the M2M System Protocol 2 Control API.

A typical server socket message sequence is illustrated in Figure 17.

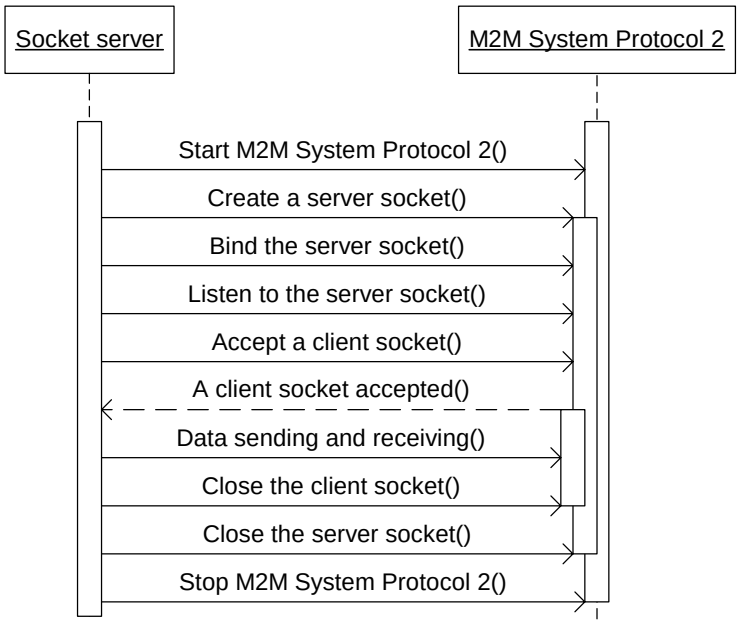
Socket server
M2M System Protocol 2
Start M2M System Protocol 2()
Create a server socket()
Bind the server socket()
Listen to the server socket()
Accept a client socket()
A client socket accepted()
Data sending and receiving()
Close the client socket()
Close the server socket()
Stop M2M System Protocol 2()

Figure 17. A typical server socket message sequence

	Note: In case the serial link between the AM and Nokia M2M module is re-established after being broken, the M2M System Protocol 2 will reset the protocol; all pending operations are interrupted, and all sockets are closed.

6.2 APPLICATION SOFTWARE EXAMPLE

In this chapter, the `Hello World` application uses the M2M System Protocol 2 client and server sockets as described in Chapter 6.1. Figure 18 shows an overview of the `Hello World` application, in which the M2M System Protocol 2 socket interface is used to send a text string "`SAY`" from the client to the server, and a text string "`HELLO WORLD!`" back from the server to the client.

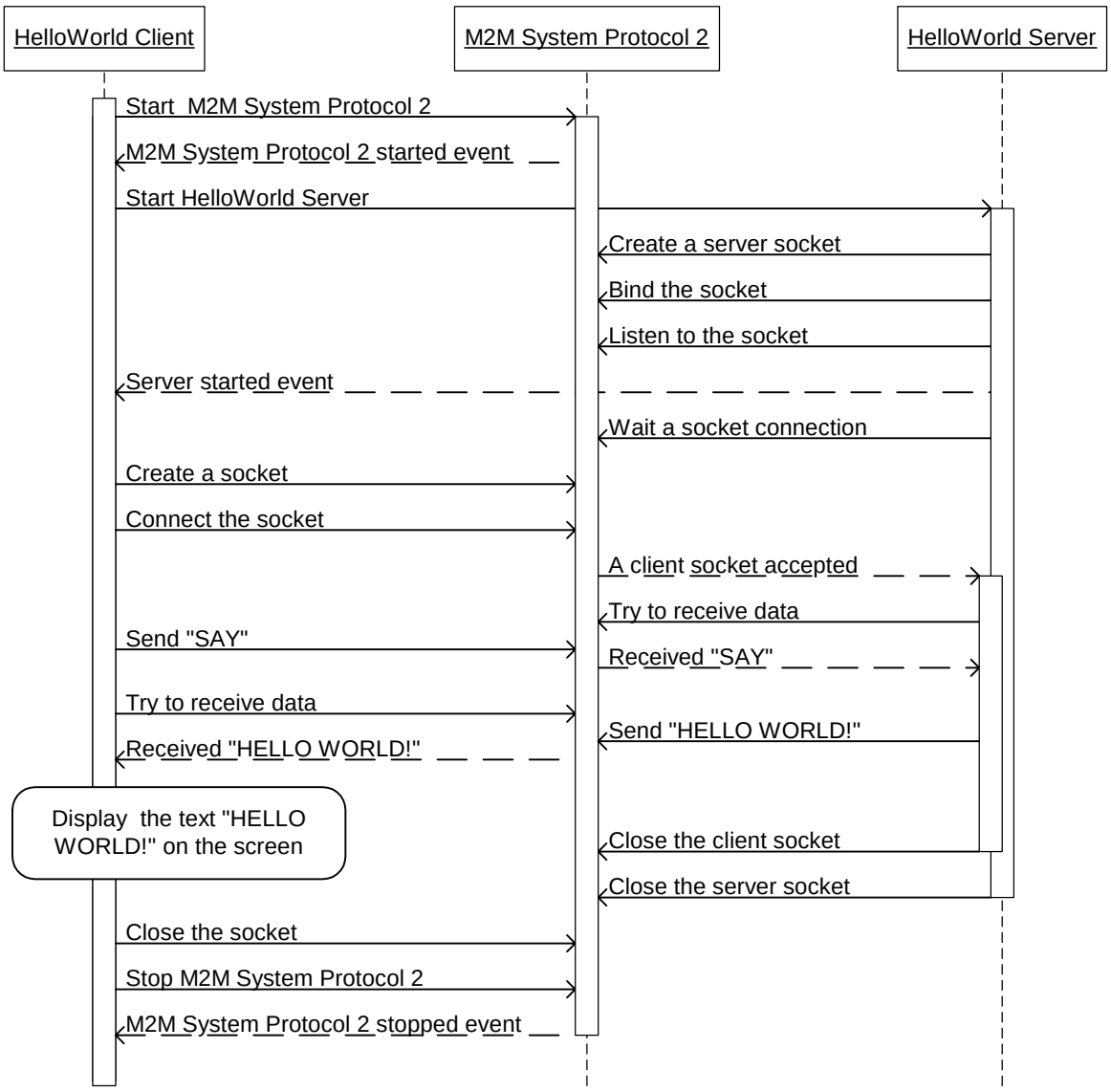

Figure 18. Overview of the Hello World application

6.2.1 Configuring and executing the Hello World application

The `Hello World` application includes both the client and server implementation. As both the client and server of the `Hello World` application use the M2M System Protocol 2 sockets, the socket peers are likely to be Nokia M2M modules. Thus, the wireless connection between the two Nokia M2M modules has to be configured so that the client can establish a connection to the server.

Note: An example configuration (for the Nokia 12 GSM module) is presented in Appendix: Nokia 12 GSM module configuration for the Hello World application.

When executing the `Hello World` application, the server must be initialized first; it must be ready to accept incoming socket connections from the client. The `Hello World` application can be executed, for example, from the command prompt.

Use the following parameters to execute the `Hello World` server:

```
> HelloWorld.exe server <com port>
```

Use the following parameters to execute the `Hello World` client:

```
> HelloWorld.exe client <com port>
```

When the `Hello World` application is executed without errors, the client will display the text "`Hello World successful!`" on the screen. If an error occurs, the client will display an error message with a failure explanation.

6.2.2 The Hello World implementation

In the `Hello World` implementation, the M2M System Protocol 2 serial link is initialized first. When the client receives a link ready indication, it tries to establish a connection to the server that is listening to incoming sockets. After the server has accepted the connection request, the client will write the text string "`SAY`" to the socket. The server reads the text from its socket and answers with a text string "`HELLO WORLD!`". Finally, the client receives the server's greeting through its socket and displays the text on the screen. Listing 1 shows the source code of the `Hello World` application.

Listing 1. Hello World – an M2M System Protocol 2 example

```
// Hello World application: HelloWorld.c

// This is an example of how to use the M2M System Protocol 2 in Win32.
// See the M2M System Protocol 2 Integrator's Manual for more
// information about this software.

// Usage: HelloWorld.exe client/server <com port>
// Expected output: "Hello World successful!" or an error message

#include <stdio.h>

// The M2M System Protocol 2 Socket interface
#include "sp2_socket.h"

// Size of buffer that holds text messages: "SAY" and "HELLO WORLD!"
#define messageBufferLen 80
static const WORD TcpPortThatServerIsListening = 0x5555;
```

```
// Client functionality
static BOOL HelloWorldClient(void);
// Server functionality
static BOOL HelloWorldServer(void);

// The function for writing a '\0' char terminated text string to a socket
static BOOL sendMessage(DWORD socket, const char *msg);
// The function for reading a '\0' char terminated string from a socket
static BOOL receiveMessage(DWORD socket, char *messageBuffer, int size);
// Cleanup and exit Hello World (just a convenience function)
static void cleanExit(int statusId, char *statusName);

// This source file has both server and client functionality.
// The first command line argument tells which role is chosen
static BOOL isClient;

// Main entry of the Hello World application
void main(int argc, char * argv[])
{
   // Serial port parameters
   BYTE COMPort;
   LONG COMPortSpeed = 115200;

   if (argc != 3)
   {
      cleanExit(0,"Usage: helloworld client/server <com port>");
   }

   if (strcmp(argv[1],"client") == 0)
   {
      isClient = TRUE;
   } else
   {
      isClient = FALSE; // i.e. isServer
   }
   COMPort = (BYTE)atoi(argv[2]);

   if (isClient)
   {
      printf("HelloWorld Client at COM port %d\r\n",COMPort);
   } else
   {
      printf("HelloWorld Server at COM port %d\r\n",COMPort);
   }


   // At first, create the M2M System Protocol 2 hLinkReadyEvent;
   // the hLinkReadyEvent is used by the M2M System Protocol 2 to
   // indicate when the link between the AM and Nokia M2M module is ready.
   if (!system_event_create(&sp2_hLinkReadyEvent) || !sp2_hLinkReadyEvent)
   {
      cleanExit(10,"ERROR: Cannot create hLinkReadyEvent");
   }

   // The hStopEvent is used by the M2M System Protocol 2 to indicate
   // when the link between the AM and Nokia M2M module is stopped.
   if (!system_event_create(&sp2_hStopEvent) || !sp2_hStopEvent)
   {
      cleanExit(20,"ERROR: Cannot create hStopEvent");
   }

   // Start the M2M System Protocol 2
   if (!sp2_start(255,10,5,COMPort,COMPortSpeed))
   {
      cleanExit(30,"ERROR: Cannot start M2M System Protocol 2");
   }
   // Wait until the M2M System Protocol 2 has started (the timeout is 3
   // seconds)
   if (system_event_wait(sp2_hLinkReadyEvent,3*1000) != SYSTEM_OK)
```

```
   {
      cleanExit(40,"ERROR: hLinkReadyEvent not OK");
   }

   // When a link is ready the protocol state must be reset.
   if (!sp2_reset())
   {
      cleanExit(50,"ERROR: Cannot reset the M2M System Protocol 2");
   }

   // Select the role: client or server.
   if (isClient)
   {
      // Server should be ready before doing client's job
      if (!HelloWorldClient())
      {
         cleanExit(60,"ERROR: HelloWorldClient failed!");
      }
   } else
   {
      if (!HelloWorldServer())
      {
         cleanExit(60,"ERROR: HelloWorldServer failed!");
      }
   }

   // Give some time for the system to send messages
   Sleep(10*1000);

   // Stop the M2M System Protocol 2
   if (!sp2_stop())
   {
      cleanExit(-10,"ERROR: Cannot stop M2M System Protocol 2!");
   }

   // The hStopEvent indicates that the M2M System Protocol 2 is stopped.
   if (system_event_wait(sp2_hStopEvent,30*1000) != SYSTEM_OK)
   {
      cleanExit(-20,"ERROR: M2M System Protocol 2 was not stopped!");
   }

   // Close all handles before exiting.
   if (!CloseHandle(sp2_hLinkReadyEvent) ||
      !CloseHandle(sp2_hStopEvent))
   {
      cleanExit(-30,"ERROR: Cannot close all handles!");
   }

   cleanExit(0,"Hello World successful!");
}

// The functionality of the Hello World client.
static BOOL HelloWorldClient(void)
{
   DWORD socket; // A client socket handle
   sp2_sockaddr_in socketAddress; // The address of the socket
   BOOL success = FALSE; // Indicates Hello World client's success/failure

   DWORD linkID; // A wireless link handle
   DWORD connectionID = SP2_DEFAULT_CONNECTION;

   // Please note that wireless link is not needed for local sockets
   if ((linkID = sp2_link_open(connectionID)) == SP2_LINK_ERROR)
   {
      printf("sp2_link_open failed\n");
      return FALSE;
   }

   // Create an M2M System Protocol 2 socket
```

```
   socket = sp2_socket(SP2_AF_INET, SP2_SOCK_STREAM, SP2_IPPROTO_TCP);

   if(socket != SP2_SOCKET_ERROR)
   {
      socketAddress.sin_family = SP2_AF_INET;
      socketAddress.sin_port = sp2_htons(TcpPortThatServerIsListening);
      socketAddress.sin_addr = sp2_inet_addr("10.0.0.1");

      if(sp2_connect(socket,(sp2_sockaddr*)&socketAddress,
         sizeof(socketAddress)) !=  SP2_SOCKET_ERROR)
      {
         const char *sayString = "SAY";
         printf("Client: send message: %s\n", sayString);
         if (sendMessage(socket, sayString))
         {
            char messageBuffer[messageBufferLen];
            if (receiveMessage(socket, messageBuffer, messageBufferLen))
            {
               printf("Client: got message: %s\n ", messageBuffer);
               if (strcmp(messageBuffer,"HELLO WORLD!") == 0)
               {
                  success = TRUE;
               }
            } else
            {
               printf ("ERROR: Client receiveMessage\n");
            }
         } else
         {
            printf ("ERROR: Client sp2_send\n");
         }
      } else
      {
         printf ("ERROR: Client sp2_connect\n");
      }

      if (sp2_close(socket) == SP2_SOCKET_ERROR)
      {
         printf ("ERROR: Client sp2_close\n");
      }
   } else
   {
      printf ("ERROR: Client sp2_socket\n");
   }

   // Remember to close the wireless link (if opened earlier)
   if (sp2_link_close(linkID) != SP2_LINK_OK)
   {
      printf ("ERROR: Client sp2_link_close\n");
   }

   return success;
}

// The functionality of the Hello World server.
static BOOL HelloWorldServer(void)
{
   DWORD serverSocket; // A server socket handle
   DWORD socket = SP2_SOCKET_ERROR; // A data socket handle from the client
   sp2_sockaddr_in socketAddress; // Socket address structure
   DWORD addrLen; // Socket address length

   // Create an M2M System Protocol 2 socket for the server to start
   // listening to client connections.
   serverSocket = sp2_socket(SP2_AF_INET,SP2_SOCK_STREAM,SP2_IPPROTO_TCP);

   if(serverSocket != SP2_SOCKET_ERROR)
   {
      socketAddress.sin_family = SP2_AF_INET;
```

```
      socketAddress.sin_port = sp2_htons(TcpPortThatServerIsListening);
      socketAddress.sin_addr = 0;

      if(sp2_bind(serverSocket, (sp2_sockaddr *)&socketAddress,
         sizeof(socketAddress)) != SP2_SOCKET_ERROR)
      {
         if(sp2_listen(serverSocket, 0x00) != SP2_SOCKET_ERROR)
         {
            // The server is now ready to receive socket connections
            addrLen = sizeof(socketAddress);
            socket = sp2_accept(serverSocket,
               (sp2_sockaddr *)&socketAddress, &addrLen);
            if(socket != SP2_SOCKET_ERROR)
            {
               char messageBuffer[messageBufferLen];
               if (receiveMessage(socket, messageBuffer,
                  messageBufferLen))
               {
                  printf("Server: got message: %s\n ", messageBuffer);
                  if (strcmp("SAY",messageBuffer) == 0)
                  {
                     const char *helloWorldString = "HELLO WORLD!";
                     printf ("Server: send message: %s\n",
                        helloWorldString);
                     if (sendMessage(socket, helloWorldString))
                     {
                        // give system time to send message
                        Sleep(3*1000);
                     } else
                     {
                        printf ("ERROR: Server sp2_send\n");
                        return FALSE;
                     }
                  }
               } else
               {
                  printf ("ERROR: Server receiveMessage\n");
                  return FALSE;
               }
               if (sp2_close(socket) == SP2_SOCKET_ERROR)
               {
                  printf ("ERROR: Server sp2_close\n");
                  return FALSE;
               }
            }
         }
         else
         {
            printf ("ERROR: Server sp2_listen\n");
            return FALSE;
         }
      }
      else
      {
         printf ("ERROR: Server sp2_bind\n");
         return FALSE;
      }
   }
   else
   {
      printf ("ERROR: Server sp2_socket\n");
      return FALSE;
   }

   if (sp2_close(serverSocket) == SP2_SOCKET_ERROR)
   {
      printf ("ERROR: Server socket sp2_close\n");
      return FALSE;
   }
```

```
   return TRUE;
}

// The function for writing a '\0' char terminated text string to a socket
BOOL sendMessage(DWORD socket, const char *msg)
{
   int bytesSent = 0;
   int size = strlen(msg)+1;
   while (bytesSent != size)
   {
      // Send data to the M2M System Protocol 2.
      // Hello World messages are very short and thus probably sent
      // in one request. However, the number of bytes written is checked
      // and sending is looped until all the data has been sent.
      int n = sp2_send(socket, msg+bytesSent, size-bytesSent, 0x00);

      if (n == SP2_SOCKET_ERROR)
      {
         printf("ERROR: sp2_send\n");
         return FALSE;
      }

      bytesSent += n;
      Yield(); // Just to let other threads run
   }
   return TRUE;
}

// The function for reading a '\0' char terminated string from a socket
BOOL receiveMessage(DWORD socket, char *messageBuffer, int size)
{
   int bytesReceived = 0;
   BOOL isMessageReceived = FALSE;

   // Read until the '\0' char is received.
   while (!isMessageReceived)
   {
      // Receive data from the M2M System Protocol 2.
      // Hello World messages are very short and thus probably received
      // in one request. However, the number of bytes read is checked
      // and receiving is looped until all the data has been received.
      int n = sp2_receive(socket, messageBuffer+bytesReceived,
         size - bytesReceived, 0x00);

      if (n == SP2_SOCKET_ERROR)
      {
         printf("ERROR: sp2_receive\n");
         return isMessageReceived;
      } else
      { // Check whether the '\0' char was received.
         int i;
         for (i=bytesReceived; i<bytesReceived+n; i++)
         {
            if (messageBuffer[i] == 0)
            {
               isMessageReceived = TRUE;
            }
         }
         bytesReceived  += n;
      }
      Yield(); // Let other threads run
   }
   return isMessageReceived;
}

// Do the cleanup and then exit.
void cleanExit(int statusId, char *statusName)
{
```

```
   printf("%s\n",statusName);

   switch (statusId)
   {
      // Successful case!
      case 0:
      break;

      // Error cases:
      case -10:
      // flow-thru
      case -20:
         CloseHandle(sp2_hLinkReadyEvent);
         CloseHandle(sp2_hStopEvent);
      // flow-thru
      case -30:
      break;

      case 60:
      // flow-thru
      case 50:
      // flow-thru
      case 40:
      // flow-thru
         if (sp2_stop())
         {
            system_event_wait(sp2_hStopEvent,10*1000);
         }
      // flow-thru
      case 30:
         system_event_destroy(sp2_hStopEvent);
      // flow-thru
      case 20:
         system_event_destroy(sp2_hLinkReadyEvent);
      // flow-thru
      case 10:
      break;

      default:
         printf("Unknown ERROR at cleanExit: %d\n",statusId);
      break;
   }

   // The statusId is passed to ExitProcess and it can be acquired by
   // using the Win32 GetExitCodeThread function (success = 0)
    ExitProcess(statusId);
}
```

To use the M2M System Protocol 2 functions from an application the M2M System Protocol 2 objects must be included within application software. Listing 2 shows a makefile template that lists the `Hello World` and M2M System Protocol 2 object files.

Listing 2. A sample makefile for the M2M System Protocol 2

```
# Makefile template for the M2M System Protocol 2 (Win32 port)

# List all the objects of the M2M System Protocol 2
# Notice that Win32-specific file names end with “win32”
SP2_OBJS = sp2_control_api.o sp2_socket_api.o sp2_link_api.o \
   sp2_common.o sp2_rs.o sp2_dll.o sp2_nll.o sp2_port_socket.o \
   sp2_port_os_win32.o sp2_port_hw_win32.o \
   sp2_system_common_win32.o sp2_dbg_wrap_win32.o

# Target ‘all’ contains application software objects instead
```

```
# of HelloWorld.o in application module software makefile
all: $(SP2_OBJS) HelloWorld.o
```

Tip: The `Hello World` makefile is a good starting point when integrating the M2M System Protocol 2 with application software.

APPENDIX: NOKIA 12 GSM MODULE CONFIGURATION FOR THE HELLO WORLD APPLICATION

This chapter provides an example of how to use the Nokia 12 Configurator to configure two Nokia 12 GSM modules as a client and a server. The configuration presented in this chapter has to be carried out in order for the `Hello World` client to establish a connection to the `Hello World` server.

The Hello World server configuration

The incoming Challenge Handshake Authentication Protocol (CHAP) for the Point-to-Point Protocol (PPP) must be configured for the server.

1. Select **Bearer Selection** from the **M2M System Mode** menu.
2. Define the incoming CHAP username and password (for example, ntn211/ntn211).

An example configuration is shown in Figure 19.

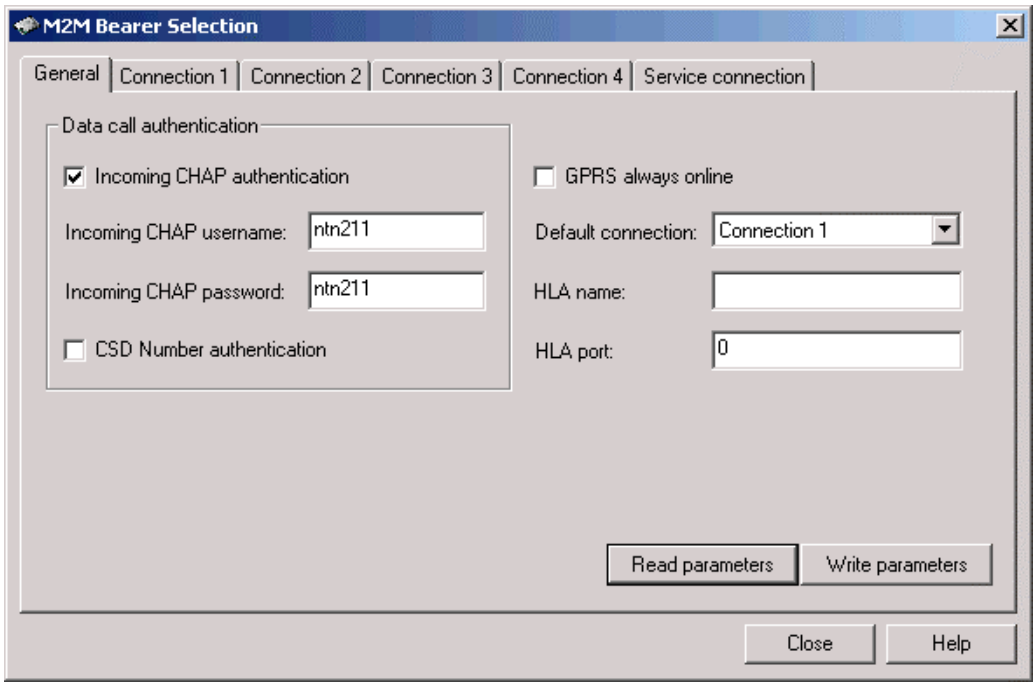

Figure 19. The Hello World server configuration for client authentication

The connection of the server must be configured so that the server is ready to accept incoming requests from the Hello World client. In this example the client will be configured so that it makes a TCP CSD call and that is why the wireless bearer of the server is configured to use TCP CSD.

3. Select **Connection 1** from the **M2M Bearer Selection** dialog.

4. Choose TCP CSD as the wireless bearer.

An example configuration is shown in Figure 20.

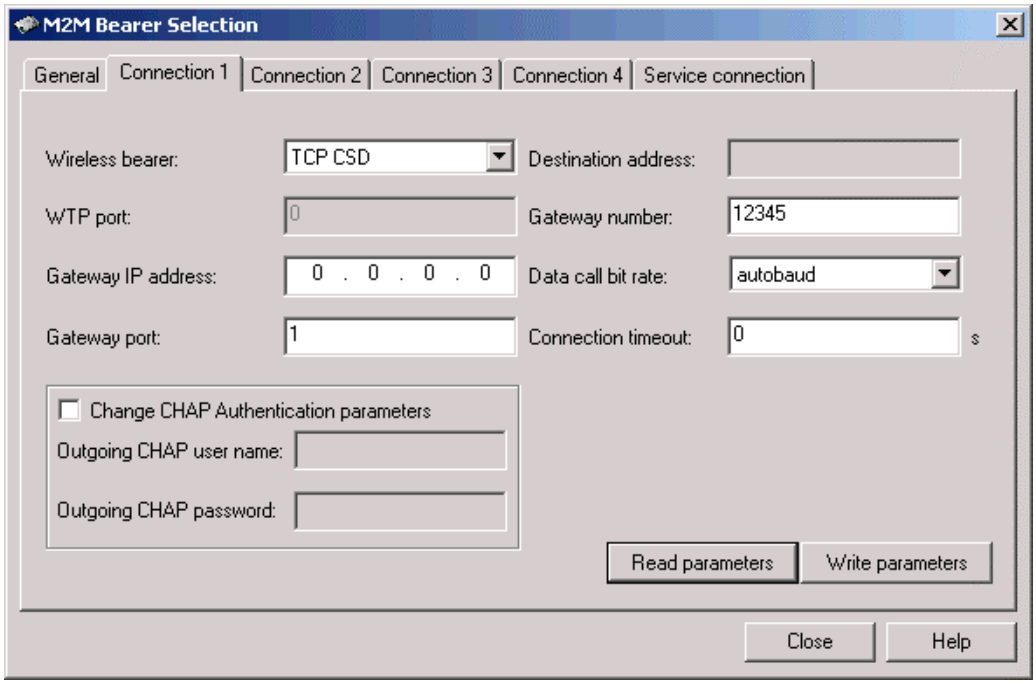

Figure 20. The Hello World server connection configuration

You also have to configure the IP address of the server. The IP address that is configured to the server must match with the IP address that the client uses as a parameter for the M2M System Protocol 2 `sp2_connect` function.

5. Select **Cable** from the **IMlet loading** menu.
6. Write 10.0.0.1 as the Nokia 12 IP address.

An example IP address configuration for the server is shown in Figure 21.

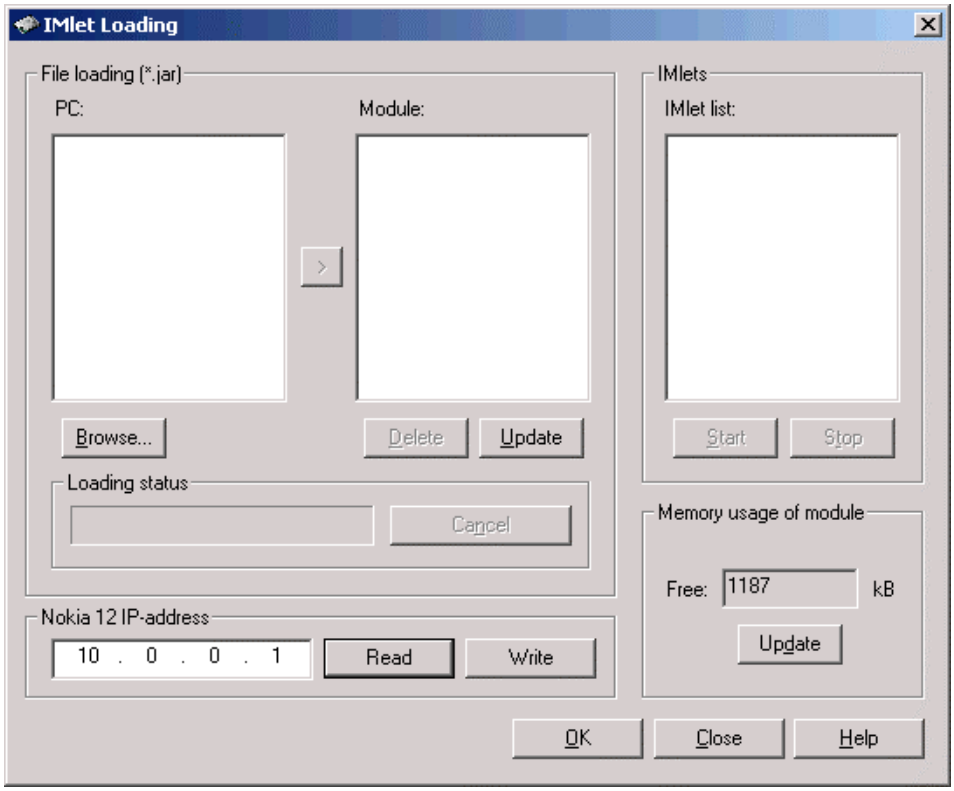

Figure 21. Hello World server IP address configuration

The Hello World client configuration

In the Hello World example, the client is configured so that it makes a direct TCP CSD call to the server. Therefore you need to configure a TCP CSD connection for the client; the client opens the default connection when it connects to the server.

1. Select **Bearer Selection** from the **M2M System Mode** menu.
2. Choose the default connection (for example Connection 1).

An example configuration is shown in Figure 22.

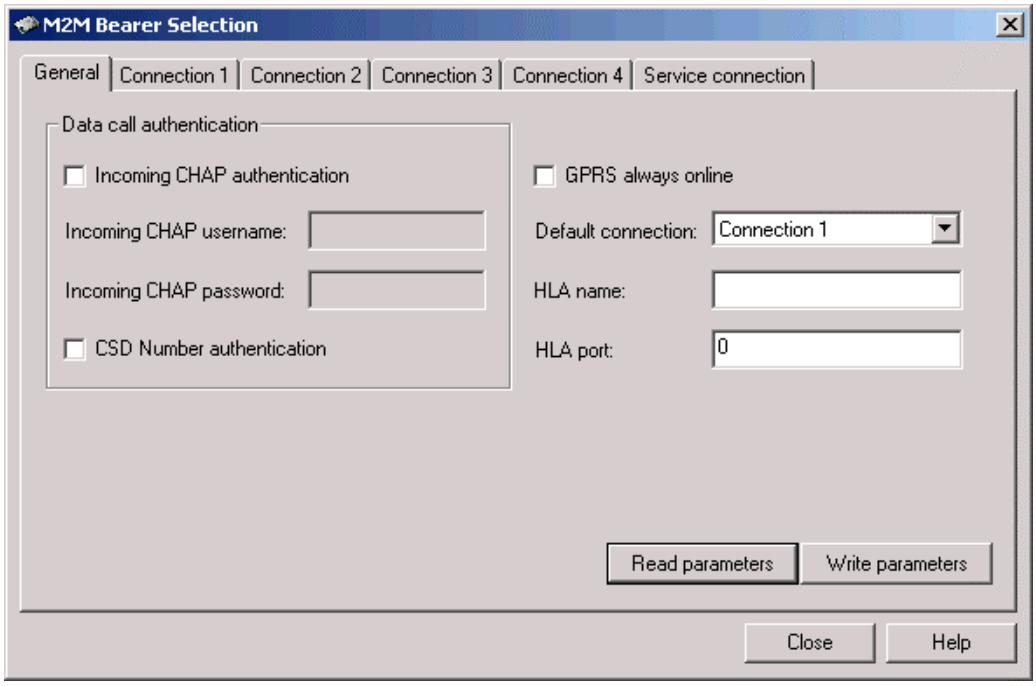

Figure 22. The Hello World client default connection configuration

A wireless connection (from the client to the server) has to be configured for the client. The outgoing CHAP configuration of the client must match with the incoming CHAP configuration of the server in order for the server to make a client authentication. In addition, the phone number of the server is defined in the 'Gateway number' field because the client calls the Gateway number to establish a data connection to the server.

3. Select **Connection 1** from the **M2M Bearer Selection** dialog.
4. Choose TCP CSD as the wireless bearer.
5. Define the outgoing CHAP username and password (for example ntn211/ntn211).
6. Define the phone number of the server (for example +123xxxxx).

A wireless connection configuration example for the client is shown in Figure 23.

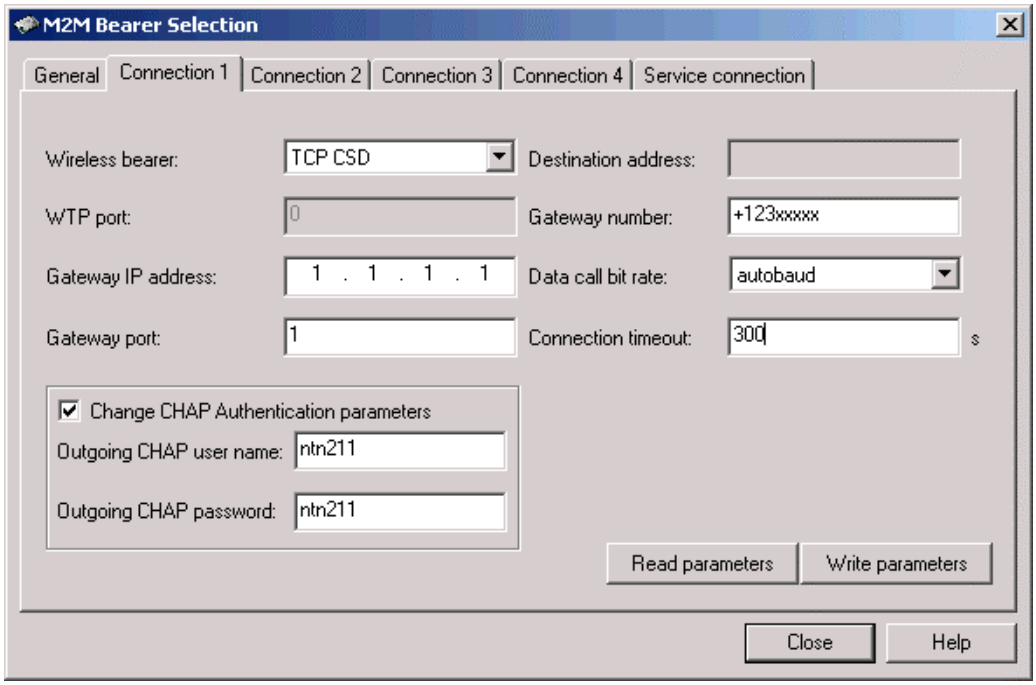

Figure 23. Hello World client configuration for a wireless connection

	Note: In this example the data call is dropped if there is a 300 second pause in the data traffic (connection timeout is set to 300 s).

And finally, you also have to configure the IP address of the client. The IP address that is configured to the client can be any valid host IP address, for example 10.0.0.2.

7. Select **Cable** from the **IMlet loading** menu.
8. Write 10.0.0.2 as the Nokia 12 IP address.

An example IP address configuration for the client is shown in Figure 24.

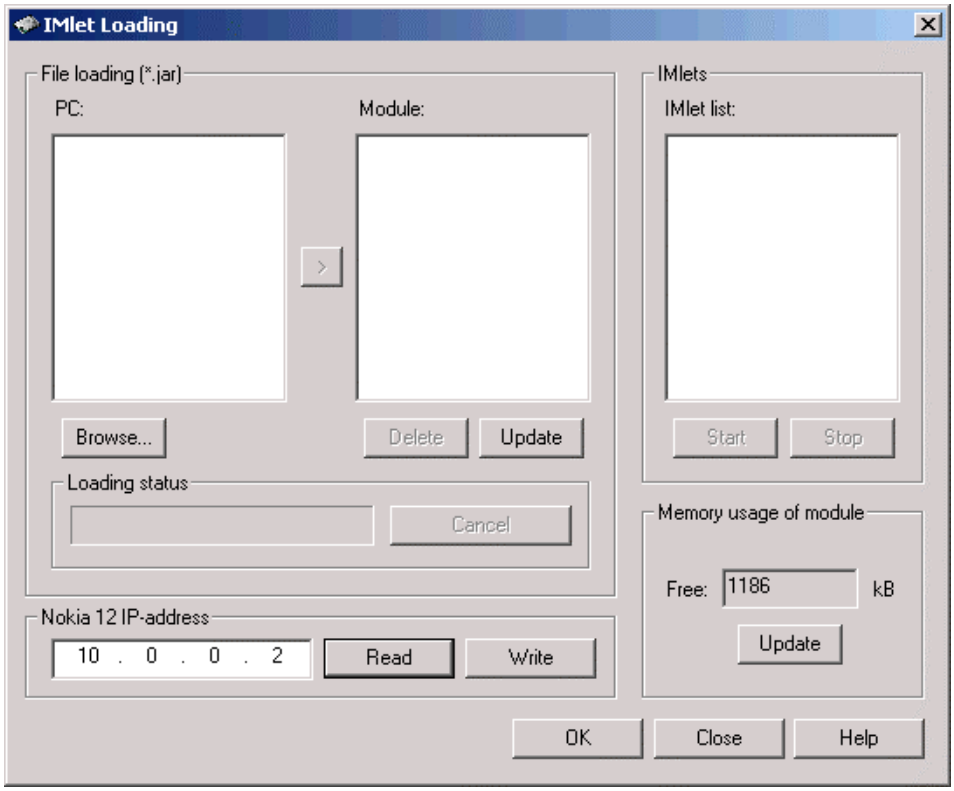

Figure 24. Hello World client IP address configuration